PGR112 – Step 12: Recap and exam

Object Oriented Programming

Bogdan Marculescu / bogdan.marculescu@kristiania.no

Agenda

- Recap
- Last year’s exam
- Exam this year

What we know

Students ...

- understands the meaning of key concepts within object-oriented programming such as class, object and method
- understands the life cycle of an object
- know what heredity and polymorphism are
- know about similarities / differences between abstract classes and interfaces
- know what exception handling entails
- has knowledge of key concepts related to code design such as connection, cohesion and encapsulation

What we know

Students ...

- understands the meaning of key concepts within object-oriented programming such as class, object and method
- understands the life cycle of an object
- know what heredity and polymorphism are
- know about similarities / differences between abstract classes and interfaces
- know what exception handling entails
- has knowledge of key concepts related to code design such as connection, cohesion and encapsulation
- **knows how to communicate with a database using an object-oriented language**
- **can explain the term "SQL injection" and how to avoid this**

Similarities / differences between abstract classes and interfaces

Abstract class	Interface
1) Abstract class can **have abstract and non-abstract** methods.	Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also.
2) Abstract class **doesn't support multiple inheritance.**	Interface **supports multiple inheritance.**
3) Abstract class **can have final, non-final, static and non-static variables.**	Interface has **only static and final variables.**
4) Abstract class **can provide the implementation of interface.**	Interface **can't provide the implementation of abstract class.**
5) The **abstract keyword** is used to declare abstract class.	The **interface keyword** is used to declare interface.
6) An **abstract class** can extend another Java class and implement multiple Java interfaces.	An **interface** can extend another Java interface only.
7) An **abstract class** can be extended using keyword "extends".	An **interface** can be implemented using keyword "implements".
8) A Java **abstract class** can have class members like private, protected, etc.	Members of a Java interface are public by default.

9)**Example:**
public abstract class Shape{
public abstract void draw();
}
	Example:
	public interface Drawable{
	void draw();
	}

javapoint

Skills

Students…

- masters a simple integrated development tool (IDE) to write and edit source code as well as compile and run simple object-oriented applications
- can define specializations of classes through inheritance
- masters the use of abstract classes and interfaces
- can apply inheritance and interfaces through code with polymorphic structure
- Can test that code works as intended

Skills

Students…

- masters a simple integrated development tool (IDE) to write and edit source code as well as compile and run simple object-oriented applications
- can define specializations of classes through inheritance
- masters the use of abstract classes and interfaces
- can apply inheritance and interfaces through code with polymorphic structure
- **can use data sources, such as files and databases, in its program code**
- can test that code works as intended

Test that the code works as required

- Create objects from classes we have created
  - With different constructors
- Call on methods we have created
  - Examine whether the result is as expected
  - Use of "debug printing"
- When something does not work as intended
  - Use debugger to find the error (s)
- Not syllabus: Automated tests

General competences

Students...

- Can explain concepts in object oriented programming, and what they entail

Packages

- Used to group classes that belong together.
- Provides unique naming.
- When we import a class (or similar), we enter both the package and the name

```
import java.awt.Color;
import java.util.*;
```

Package java.awt

Class Color

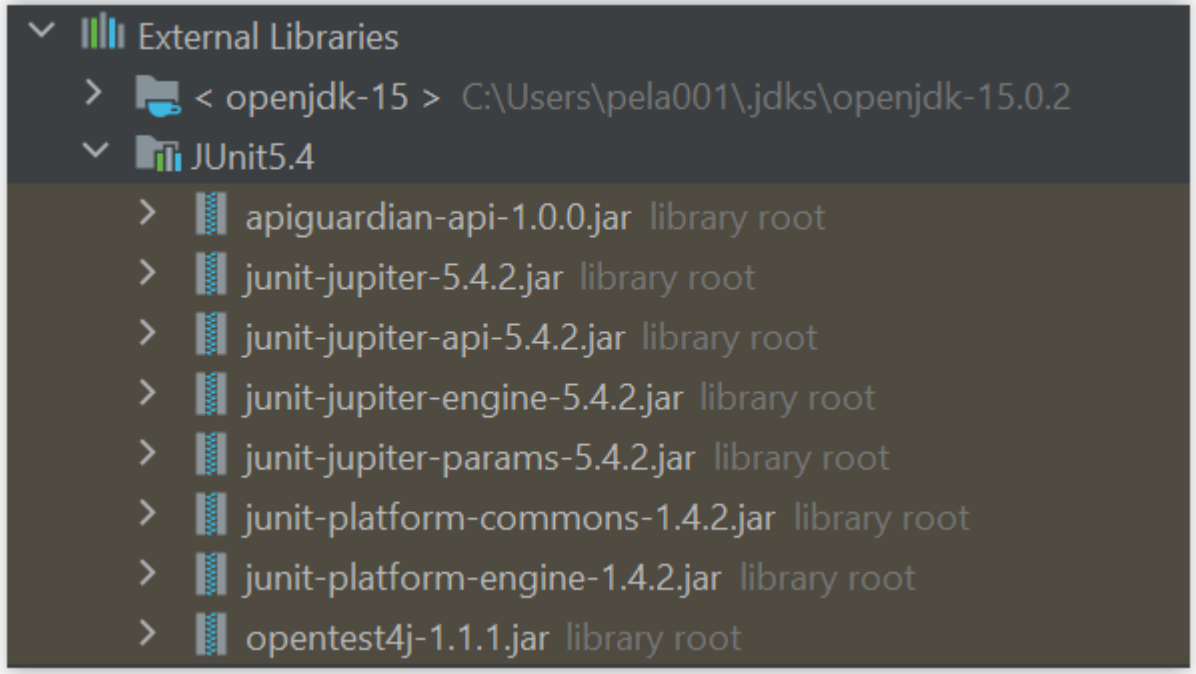
java.lang.Object
    java.awt.Color

So why do we not import String?

- Anyone wondered about that?
- String is in the java.lang package.
- We always have that package available, and do not need to import ☺

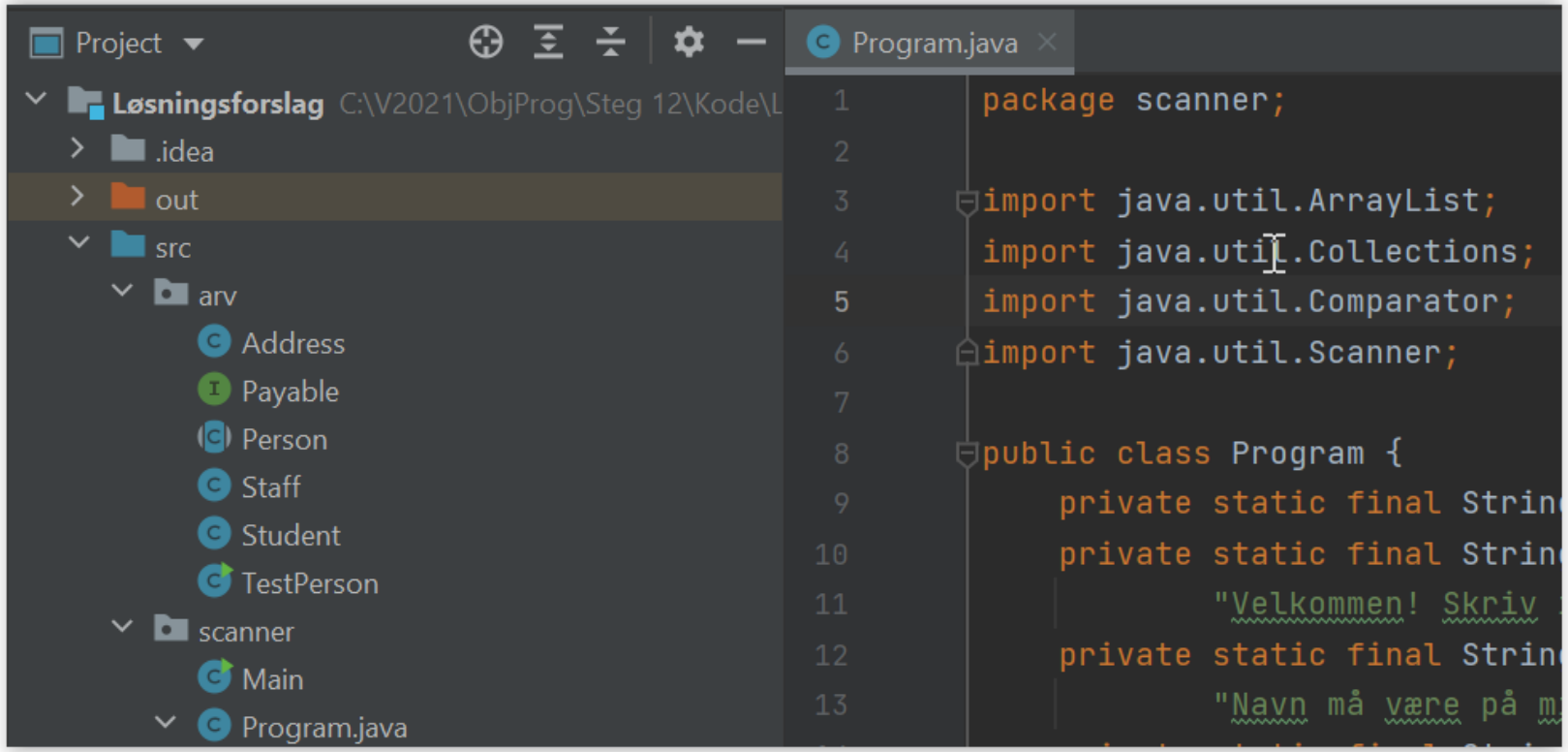
Packages outside the standard Java libraries

- We can choose to use packages outside the standard Java libraries
- We already did: Junit 5 – used for running our tests



Create your own packages

- When a project grows in size, it becomes clearer if we put the files in packages. Which package the file belongs to is described at the top of the file (before any imports)



Naming of packages

- If a package is intended to be used outside the project, then the naming becomes important.
- The naming convention described by Oracle states that companies should use the reverse internet domain as a starting point for their packages.
- Company example.com -> com.example.package

- As we work with "small" programs in this subject, packages are not very important. You can read more here.

Packages and permissions

Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

Trouble using my code starting point?

- When using existing code (e.g. Starting point code): ensure proper import of packages.
- This applies to your own code too
- Remember: 2 or more classes may have the same name, but only in different packages

Last year’s exam

- Read from file
- It is planned that inheritance will be used for the objects that are created
- Functionality for extracting information about the objects
- In other words, similar topics as for the assignment in week 11. But the assignment was be smaller.

Last year’s exam

- What if you’re unable to read from file??
- Plan B: Then you create the objects yourselves, and work from there.

Questions and answers

Høyskolen Kristiania